O que significa o termo "guloso" em um algoritmo guloso?

O termo **"guloso" (greedy)** descreve um tipo de **estratégia de otimização** usada por alguns algoritmos. A ideia central de um algoritmo guloso é:

"Em cada passo, escolha a melhor opção local possível, esperando que isso leve à melhor solução global."

Ou seja, o algoritmo toma **decisões locais ótimas** a cada etapa, **sem se preocupar com as consequências futuras**. Ele é **"guloso"** porque **pega o que parece melhor na hora**, como alguém faminto que escolhe a comida mais saborosa sem pensar na dieta.

Exemplo simples:
Suponha que você queira dar o **troco em moedas** com o menor número de moedas possível. Se você sempre escolher a moeda de maior valor que cabe no troco restante, isso é um comportamento **guloso**.

Isso é um tipo de critério de otimização?

Sim! A abordagem gulosa é **uma heurística de otimização** — ela visa **encontrar uma solução ótima ou próxima disso**, mas **não garante sempre a melhor solução global**.
Para certos problemas (como o da **mochila fracionária**, por exemplo), o algoritmo guloso **encontra a melhor solução**.

Qual o critério do algoritmo guloso deve ser utilizado no problema da Mochila Fracionária?

Deve-se primeiro encontrar o **valor unitário** (valorTotal / pesoTotal) de cada um dos itens. Em seguida, **priorizamos a inserção dos itens que possuem o maior valor unitário**.

Análise de complexidade – Mochila Fracionária

Considerando o código do repositório:

https://github.com/EvandroFerraz/20251_maua_cic401.git

Etapa	**Complexidade**
Leitura dos dados e criação da lista de n itens	**O(n)**
Cálculo de valor unitário (feito no construtor)	**O(n)**
Ordenação com Collections.sort()	**O(n log n)**
Iteração para preencher a mochila	**O(n)**

Collections.sort() usa o algoritmo **TimSort**,

TimSort é um algoritmo híbrido baseado em:

- Merge Sort

- Insertion Sort

Foi projetado para aproveitar padrões naturais de ordenação que já possam existir na lista, tornando-o muito eficiente na prática. Segundo a documentação, possui uma complexidade **O(n log n)**.

Dessa forma, a complexidade total do algoritmo utilizado na resolução do exercício 4 seria: **3*n + n*logn**. Desconsiderando constantes e mantendo só o termo dominante temos uma complexidade de **O(n log n).**